Naval Research Laboratory
Washington, DC 20375-5320

NRL/MR/5530--94-7464

A Software Architecture for Adding New Interaction Techniques to a Command and Control Based Testbed

JAMES N. TEMPLEMAN
DEBORAH HIX
ROBERT J. K. JACOB

Human-Computer Interaction Laboratory
Information Technology Division

March 25, 1994

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 25, 1994	Interim

4. TITLE AND SUBTITLE

A Software Architecture for Adding New Interaction Techniques to a Command and Control Based Testbed

5. FUNDING NUMBERS

PE - 62234N

6. AUTHOR(S)

James N. Templeman, Deborah Hix, and Robert J. K. Jacob

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/MR/5530--94-7464

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

The Human-Computer Interaciton (HCI) Laboratory at the Naval Research Laboratory (NRL) is developing high-performance interactive computer systems for use in Naval command and control applications. New technology allows computerized systems to make greater use of a person's natural physical, perceptual, and cognitive skills. These systems make it faster and more straightforward for users to access and enter information within the context of time critical, real world situations.

Our work focuses on developing novel interaction techniques—ways of using physical input and output devices to perform user tasks. Previous work has demonstrated the effectiveness of individual interaction techniques. This study is now being extended to encompass multi-modal interaction, in which combinations of interaction techniques are available to a user. We are incorporating these techniques into a command and control testbed, that will be tested through empirical, user-based evaluation.

Such new capabilities increase the complexity of software development and maintenance requirements of already complex systems. New software architectures must co-evolve with the technology to structure and manage these additional requirements. This report describes the new demands placed on interactive systems and explains how we are addressing this challenge, through development of a flexible software architecture.

14. SUBJECT TERMS

Human-computer interaction
Interaction techniques
User interface management systems
Software architecture
Command and control

15. NUMBER OF PAGES

33

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

CONTENTS

A SOFTWARE ARCHITECTURE FOR ADDING NEW INTERACTION TECHNIQUES TO A COMMAND AND CONTROL BASED TESTBED

1. INTRODUCTION

At the Human-Computer Interaction (HCI) Laboratory at the Naval Research Laboratory (NRL), we are developing new interaction techniques using input devices to track a user's eye, head, and positional hand movements as a means for communicating with an interactive computer system. These techniques open up a new set of possibilities for making user interactions with computers more natural and efficient. An interaction technique is a way of using physical input and output devices to perform useful tasks. These are typically *low-level physical user tasks*, like selecting or moving an item on the screen. An interaction technique couples a user's actions with visual and possibly auditory or tactile feedback to allow the user to access, enter, or manipulate information. Interaction techniques include such media as 3D motion, virtual world interaction, gesture input, tactile feedback, speech, non-speech audio, and eye movements. Use of these media in various combinations with each other in a human-computer interface provides *multi-modal interaction techniques* for users.

It might seem odd at first to think of using head and eye movements – the same actions that people use primarily to alter their view of the world – to provide input to control a computer. But there are ways to exploit and amplify the way people normally use their visual systems. (Examples are given in Section 2 of this report.) The more natural an interaction between user and computer, the less learning is required, and the less a user has to remember in order to use the technique. Minimizing the conscious effort required to perform an application task may allow a user to perform that task more quickly and relieve the user of the cognitive load associated with carrying out artificial steps to achieve a goal. Our philosophy is to use each technique in the most natural way, and then to combine different techniques to allow a user to perform more complex operations.

High-bandwidth, multi-modal human-computer interaction techniques are a promising solution to the Navy's need for higher-performance communication between users and computers, particularly in military systems in which user response speed is critical. Such interaction techniques can improve operator-to-computer communication in a command and control system, making computer use faster and more natural for military decision makers, and enabling a commander to speed up information gathering and decision making processes during a crisis.

However, these new interaction techniques must be evaluated to see if they actually improve user performance over existing techniques. Initially, this can be done at a very low level. For example, in a related 6.1 project (entitled Dialog Interaction Techniques) we are in the process of measuring how much faster it is to select an object with the eye than with a mouse. There are many different factors that contribute to the usefulness of an interaction technique; the speed with which a user can perform tasks with a technique is not the only consideration. Interaction

Manuscript approved January 13, 1994.

techniques can also help a user form a better understanding of a situation or reduce mental workload in a crisis.

To discover how useful a new technique really is in practice, it needs to be placed within the context of a meaningful operation. Previous work at the HCI Lab has used low fidelity applications emphasizing generic user tasks, largely for concept-proving. We now intend to transition into more realistic Naval-related applications. In particular, we are developing a project testbed based on Naval command and control systems (C^2 systems) as a vehicle for this research (Hix 1993). The testbed will incorporate new interaction techniques both in isolation and in combination (i.e., affording multi-modal communication), and will be used for empirical evaluation of these techniques in human-computer interfaces.

Development, evaluation, and application of high-performance interaction techniques place new requirements on user interface software architectures and tools, which will require advances in their technology. Presently, high-performance or non-traditional interaction techniques are typically built in ad hoc ways, using little or no underlying methodology, and have almost no development support tools. Current user interface software technology is behind the state-of-the-art in interface design. User interface management systems, or UIMSs, (discussed in Section 2.3) for supporting higher-performance interaction do not exist because the basic principles needed for building such complex systems are not yet known.

We are investigating the software architectures and tools necessary to produce, study, and apply multi-modal interaction techniques. Our starting point for this work is a specification technique for developing interactive computer systems based on concurrent state transition diagrams and its related software, developed at NRL (Jacob 1985, 1986). It will serve as the basis of the command-and-control-based project testbed that will be used to build prototypes of new high-performance interaction techniques, and will be extended and modified as needed to support new interaction techniques and media.

This report describes the software architecture for the project testbed and the design of its interaction object-based UIMS. It also describes some prototype interaction techniques we have implemented on this testbed and their software requirements: a natural head control for zooming and panning a command and control map display, and the use of eye movements for selecting displayed objects.

2. Systems Under Development at the HCI Lab

One of our research goals at the HCI Lab is to provide ways of enhancing a Naval operator's use of information systems. Modern sensor and signal processing systems allow tracking of the direction of a user's gaze, and the position of a user's head. We have developed new interaction techniques based on these additional channels of user input, and want to determine if they are easy and effective to use.

It is one thing to devise a new interaction technique which seems to have advantages over existing techniques; it is another to empirically verify the practical advantages (e.g., improved user performance) of one technique over another. Our C^2-like testbed will provide a context for evaluating and comparing existing and new techniques. It is important to note that the purpose of this testbed is not to produce a high-fidelity C^2 simulator. Rather, the purpose is to produce a command-and-control-oriented testbed as a foundation for developing and empirically evaluating interaction techniques. This empirical focus is a unique aspect of our work; few other HCI research groups perform empirical experiments to evaluate their work, especially for new interaction techniques.

2.1. Interaction Techniques

2.1.1. Pre-Screen Projection

Our view of the world changes whenever we move our head. We are familiar with the way our view changes as we move about, and control our view by positioning our head. By tracking a person's head and making the viewpoint used to compute a projected scene correspond to the person's physical viewpoint, head motion can be used to control the view of synthetic imagery.

Based on this idea, we are developing a new interaction technique called *pre-screen projection* (Templeman 1993a, 1993b). It works by tracking a person's head and then altering the view presented on the screen (see Figure 1). By making the virtual and physical viewpoints correspond, images presented on the screen match those that would be seen if the object existed in space in front of the user. This carries many of the natural dynamic properties of the visual world over to computer-generated imagery, to make the view react in a way consistent with the user's expectations, that is, in a natural way. A virtual object must be in front of the screen for its projected screen image to expand as a user moves toward it.

When the entire scene consists of a single flat picture in front of the screen, head movements provide a natural, hands-free way to pan-and-zoom over it. A user pans an image on the screen by simply moving their head from side to side and zooms in and out by simply moving toward or away from the screen.

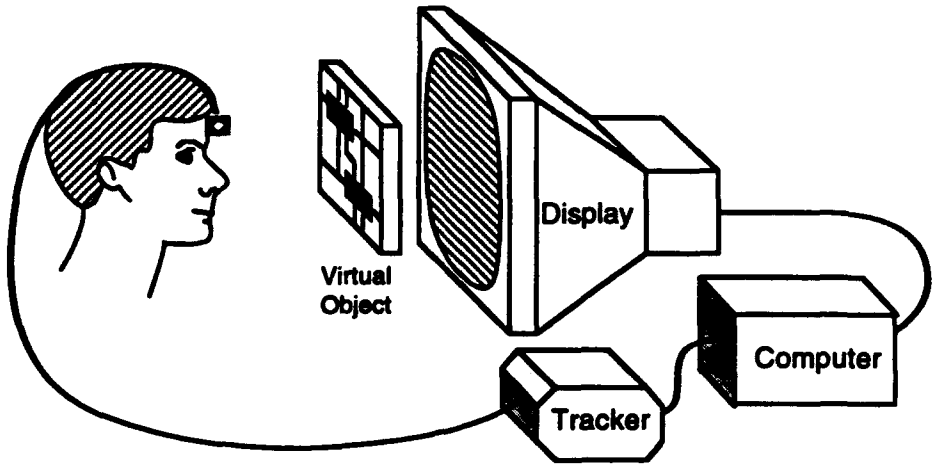

FIGURE 1: PROJECTING AN OBJECT IN FRONT OF THE SCREEN.

Pre-screen projection provides continuous visual feedback that guides the user in selecting a view. The user simply moves their head toward a point of interest to acquire the desired view. This technique gives the user access to a virtual viewing surface much larger than the display screen while leaving the hands free to perform other operations. Three dimensional features may be added to strengthen the effect of a dynamic view, which appears in a frame of reference that remains fixed in three-dimensional space.

This new technique for panning and zooming over a map is well suited for performing the kinds of spatial resource management carried out in C^2 applications. Once fully developed, this technique should, for example, allow analysts to get more use out of situational displays presented on desktop or large screen displays.

This new technique provides an interesting contrast to the eye-gaze driven interface previously developed at NRL's HCI Lab (discussed in Section 2.1.2). Pre-screen projection provides a continuous stream of output in response to a user's head movements. The strong dynamic graphical feedback component contrasts sharply with the eye tracking interface, where only minimal changes to the screen are made in response to eye movements, to avoid drawing the eye's attention and thus setting up an undesirable feedback loop.

2.1.1.1. Implementation

The initial version of pre-screen projection uses an Ascension Bird™ device to track the head, and a Silicon Graphics Iris 4D/210VGX workstation to generate the interactive view. A monocular (non-stereo) view is presented. The scene, a simple arrangement of ships and planes drawn on top

of a grid, is viewed from above, with the virtual surface of the ocean falling two feet in front of the screen.

Pre-screen projection is straightforward to implement. All that is required is to map the computational viewpoint and the user's physical viewpoint into the same coordinate system, and define the viewing volume so that its edges line up with the window on the screen. The apex of the viewing pyramid moves in concert with the physical viewpoint. The graphics system must redraw the scene fast enough to produce smooth animation. To draw objects in front of the screen, simply place them there in the model coordinate system, and make sure that the near clipping plane falls in front of them. Deering (1992) describes in detail how to register the virtual scene precisely in the user's view.

2.1.2. Eye Tracking

Eye movements provide a means by which users and computers can communicate information. That is, the computer will identify the point on its display screen at which the user is looking and use that information as a part of its dialogue with the user (Jacob 1990). (Note: Throughout this paper, the word "dialogue" simply refers to the interaction between user and computer.) For example, if a display showed several icons, a user might request additional information about one of them. Instead of requiring the user to indicate which icon was desired by pointing at it with a mouse or by entering its identifier with a keyboard, the computer can determine which icon the user is looking at and immediately give information on it. The user's actions (i.e., looking at an item) are the same as in the real world, but the physical effect is different in the computer world than in the real world.

Eye trackers have existed for a number of years, but their use has largely been confined to data collection for post hoc analysis. The equipment is now becoming sufficiently robust and inexpensive to consider its use in a real-time user-computer interface. What is now needed to make this happen is appropriate interaction techniques that incorporate eye movements into user-computer interfaces in a convenient and natural way.

A user interface based on eye movement inputs has the potential for faster and more immediate interaction than current interfaces, because people move their eyes extremely rapidly and with little conscious effort. A simple thought experiment suggests this speed advantage: Before operating any mechanical pointing device (i.e., a mouse), a person usually looks at the destination to which they wish to move the mouse. Thus the eye movement is available as an indication of the person's goal before they could physically actuate any other input device.

However, people are not accustomed to operating devices in the world simply by moving their eyes. Our experience is that, at first, it is empowering for a user to be able simply to look at what

they want and have it happen, rather than having to look at it and then point and click it with the mouse. Before long, though, it becomes like the "Midas Touch". Everywhere the user looks, another command is activated; the user cannot look anywhere without issuing a command. The challenge in building a useful eye movement interface is to avoid this Midas Touch problem. Carefully designed new interaction techniques are thus necessary to ensure that they are not only fast but that they use eye input in a natural and unobtrusive way. Our approach is to think of eye position more as a piece of information available to a user-computer interaction involving a variety of input devices than as a general purpose input device used in isolation.

A further problem arises because people do not normally move their eyes in the same slow and deliberate way they operate conventional computer input devices. Eyes continually dart from point to point, in rapid and sudden saccades. Even when a user thinks they are viewing a single object, the eyes do not remain still for long. It would therefore be inappropriate simply to substitute an eye tracker as a direct replacement for a mouse. Wherever possible, we therefore attempt to obtain information from natural movements of the user's eye while they view the display, rather than requiring the user to make specific trained eye movements to actuate the system. In our design, the computer constantly responds to the user's gaze by determining the visual object nearest to where the user is looking, and displaying additional information about that visual object. The added information is presented in a status window at the side of the main window. This contrasts with the way mice are typically used to explicitly select a target, by precisely positioning a cursor over the visual item of interest, and clicking a mouse button to initiate an action (e.g., displaying additional information about that visual item).

2.1.2.1. Implementation

Our solution to the problem of using eye movement data is to partition it into two stages (Jacob 1991). First the system processes raw data from the eye tracker in order to filter noise, recognize fixations, compensate for local calibration errors, and determine, from available information, where a user is looking on the screen. This processing stage uses a model of eye motions (fixations separated by saccades) to drive a fixation recognition algorithm that converts the continuous, somewhat noisy stream of raw eye position reports into discrete tokens that represent a user's intentional fixations. Then these tokens are passed to a user interface management system (UIMS), along with tokens generated by the low-level input from other devices being used simultaneously, such as a keyboard or mouse.

Output of the recognition algorithm is converted to a stream of tokens, in order to make the eye tracker data more tractable for use as input to an interactive user interface. The system reports tokens for eye events considered meaningful to the user-computer interaction, analogous to the

way that raw input from a keyboard (e.g., shift key was depressed, letter 'A' key was depressed, etc.) is turned into meaningful events (e.g., one ASCII upper case 'A' was typed). Tokens are reported for the start, continuation (every 50 milliseconds, in case the system is waiting to respond to a fixation of a certain duration), and end of each detected fixation. Each such token is tagged with the actual fixation duration, so an interaction technique that expects a fixation of a particular length will not be skewed by delays in processing by the UIMS or by the delay inherent in the fixation recognition algorithm. Between fixations, a non-fixation token is reported periodically by the system, indicating where the eye is. However, our current interaction techniques ignore this token in preference to the fixation tokens, which are more filtered. A token is also reported whenever the eye tracker fails to determine eye position for 200 milliseconds and again when it resumes tracking. In addition, tokens are generated whenever a new fixation enters or exits a monitored region, just as is done for the mouse. Note that jitter during a single fixation will never cause such an enter or exit token, since the nominal position of a fixation is determined at the start of a fixation and never changes during the fixation. Jitter is ignored between fixations. These tokens, having been processed by the algorithms just described, are suitable for use in user-computer interaction in the same way as tokens generated by mouse or keyboard events.

2.2. An Application Testbed for Interaction Techniques

In developing our C^2-like testbed, our first step was to study existing command and control systems (Hix 1993). Our goal is to identify tasks performed in current C^2 systems, so the testbed for multi-modal interaction can focus on the most critical aspects of command and control. We have visited a number of local facilities to meet with people developing and using modern C^2 systems, and received a number of demonstrations.

2.2.1. Motivation for the C^2 Testbed

As discussed earlier, the main purpose of the C^2 testbed is to support creation and empirical evaluation of interaction techniques, both singly (one technique used in isolation) and in combination. Earlier research in the HCI Lab on new interaction techniques has used rather simple, often non-military domains and tasks (Jacob & Sibert 1992). This research has yielded valuable information about alternative interaction techniques such as eye gaze and three-dimensional trackers. Continued work in this area should have even greater value and long-term consequences for Naval C^2 systems if it is set in the context of a more realistic, command-and-control-like testbed to be used for experimentation.

Many of the techniques being considered for inclusion in the testbed are unusual, non-routine techniques (such as eye gaze, head mounted trackers, pen-based and other gestural input, voice,

and so forth). Obviously, these techniques are, in some cases, drastically different in look, feel, and behavior from the standard mouse-and-keyboard interfaces that users most commonly use. As a result, people often mistakenly assume that the very uniqueness and novelty of such techniques make them naturally "better" – or at least more fun – for users. This is, of course, not necessarily the case. Thus, rather than assuming that exciting new technology improves user task performance and satisfaction, we want to determine empirically whether it does or not, and in which cases and for what tasks. So a goal of using the C^2 testbed for experimentation is to evaluate user performance. Testbed development is also extending existing work at the HCI Lab on architectures for user interface software management.

2.2.2. *Requirements for Testbed Development*

There are several critical requirements for the testbed:

- The testbed needs to capture essential tasks performed in a meaningful application – namely, C^2.

Essential tasks are those that are most frequently performed by a user, those that are mission critical, and so on.

- The testbed should be fairly easy and quick to develop.

We are studying new interaction techniques within a C^2 framework, not trying to develop a superior C^2 simulator. We are not in the business of developing C^2 systems, and unfortunately for us, those who are in that business have not built them with user interfaces extensible enough to accept novel interaction techniques.

- The testbed should be easy for a novice (someone not skilled or knowledgeable in C^2 systems) to learn to use.

The testbed needs to capture the essence of C^2 without requiring a user to know a lot about modern warfare.

- The testbed should be extensible.

New tasks, operations, modes, and of course devices, should be easy to add to evaluate new techniques. Incremental design allows us to accommodate anticipated future expansion of the testbed.

- We need control over internal connections between the testbed application and the interaction techniques.

It is not always feasible to simply substitute a new device or technique for the mouse in an existing application. Different interaction techniques are appropriate for different situations and different types and levels of user tasks, and these techniques are not necessarily interchangeable. There is a

very close coupling between displayed graphics and the input device(s) used to interact with and manipulate those graphics, in a tight feedback loop. Visual changes on the screen provide appropriate feedback to a user's actions. Specification of feedback for those actions defines an interaction technique. At a more global yet subtle level, layout of graphics on the screen both affords and constrains how a user operates on the graphics. Two-dimensional window-based layouts typically use a mouse for user interaction, and are designed around the capabilities and limitations of a mouse. In a very real sense, the nature of an input device influences the screen display. Different input devices, like an eye tracker or a 3D tracker, will impose their own demands on how the screen is used. This raises the question of how best to use the display when the user has simultaneous control over multiple input devices. This is a key research issue we are addressing. There are also difficult technical problems involved with adding a new device or technique into a system that was not designed to be flexible enough to deal with it. This is addressed in more detail in Section 3.2. under windowing systems and graphics tools.

2.2.3. Approach to Developing C^2 Testbed

These requirements all argue against adopting an existing, sophisticated, complex C^2 system as the basis for our testbed for evaluation. These requirements are a key in driving development of our C^2 testbed. Specifically, we will use an incremental approach to producing the C^2-like testbed, incorporating scenarios that will allow users to perform different kinds of C^2 tasks. The testbed will focus on generic, commonly performed tasks. The testbed architecture will be such that it can support the addition of new interaction techniques and devices to the testbed as "snap-ons", so that new techniques can be incorporated as quickly and easily as possible. This *generic, extensible testbed framework* will provide us with a suite of user tasks for a specific application area, namely C^2. It will include a variety of user tasks at all levels dealing with mission planning. For example, a high-level task might be "assess battle damage", while a low-level task might be "assess battle damage on a specific ship". The suite of common tasks is representative of *what* tasks C^2 systems support; the interaction techniques relate to *how* the tasks are performed.

Some of the interesting issues involved in developing the testbed include how to design the human-computer interface incorporating new interaction techniques so that they can effectively be compared to current techniques (e.g., mouse, keyboard). It is overly simplistic and optimistic to assume that an eye gaze device for a selection task substitutes directly for a mouse for that same task. Similarly, it would be possible to develop a testbed scenario using the eye gaze technique that is inherently better than a mouse-driven version of the same scenario, and vice versa. In this case, all comparative performance measures are obviously fallacious.

The basis on which we will choose interaction techniques for particular tasks, another interesting issue in testbed development, is not yet formalized. Obviously some interaction techniques work better for some types of tasks than others. A simple example is the awkwardness of entering alphanumeric characters one at a time by clicking on a keyboard display on the screen. A mouse simply does not work well for entering discrete alphanumeric characters; the traditional keyboard is much better. So in determining which techniques best fit which tasks, we will base our hypotheses on any existing work on similar interaction techniques, naturalness, our own expertise, and some pre-testing of interaction techniques on simple tasks.

Another issue is the criteria for choosing initial tasks for the testbed. The criteria are fairly obvious; namely, the tasks most often performed and the tasks that are most critical to accomplishing a mission. Tasks must also be designed and implemented in a reasonable time, so they cannot be tremendously complex. They must also be learned by users in a reasonable time, since we will not be able to obtain users as subjects in experiments for large amounts of time. (This latter criterion, is, of course, counter to real C^2 systems, which can have 26 weeks of training.)

Some other technical issues in testbed development involve simulation in the testbed; it must allow the user to dynamically adapt a plan if unforeseen events occur, and to dynamically develop new plans to capitalize on occurring events. However, supporting this kind of environment is a difficult technical challenge. Also, a database of Naval information (e.g., about ships, weapons, personnel, and so on) must underlie the testbed.

2.2.4. Philosophy Behind Our C^2 Testbed

The underlying philosophy for the first versions of the testbed, as mentioned previously, will be to "keep it simple," for the following reasons:

- Real C^2 systems are quite complex, and many diverse systems exist. There is little point in expending energy in reinventing the wheel. It would be nice if our work inspired someone with an existing C^2 system (and the requisite expertise) to add our interaction techniques to their system.
- Both the eye tracking and head tracking interaction techniques appear best suited for enhancing a user's ability to readily access information. We expect these techniques to minimize unnecessary actions, and help a user maintain orientation within a geographic context. Situational awareness is central to the performance of someone using a C^2 system.
- There is a limit to how much a novice user will be able to learn about the system before that user can perform a meaningful task, suitable for measuring performance. Most "users" that we will have access to for evaluating the interaction techniques in our C^2 testbed will not be aware of the inherent capabilities of an F-111 bomber, for example. To that end, the testbed will provide

them with "pre-digested" descriptive information about each of their resources and any other appropriate objects in a particular scene. Each weapon system, for example, might have a numeric rating of its destructive capacity (for appropriate targets), and each target would have a corresponding rating of its vulnerability to weapon hits.

We will use simplified, mock data instead of real military databases. These can be garnered from openly published war games, like Harpoon, as well as from readily available sources such as Jane's and Defense Mapping Agency data.

2.2.5. *Ideas for the Initial Testbed*

In the early stages of our testbed development, we are most interested in interaction techniques for input and their effects on output (presentation). The planning task is largely non-numeric and temporal, with graphical and visual needs. In addition to its output requirements, planning occurs through a great deal of interaction of the user with the C^2 system. Thus, planning is a logical high-level task for us to develop, at least initially, in our C^2 testbed.

For the reasons just explained, the scenarios and tasks we intend to develop will focus primarily on mission planning tasks. Accordingly, the initial testbed will focus on the planning process for an air-strike scenario, which can be broken down as follows:

Situation Assessment

-> Target Selection

-> Asset Allocation and Weapon Load-out

Use of a static C^2 testbed will be very effective in our early work. For example, a map showing geographic deployment of friendly and enemy resources can allow a user to study aircraft available on each carrier and air-base, and candidate enemy targets. The user might then specify weapons load-out of the aircraft, group aircraft into functional units, and assign these units to enemy targets. The user can acquire the necessary information from databases accessible through the testbed, and will not be expected to know such information *a priori*. This kind of initial planning, which can be done without a simulation module in the testbed, is critical to the success of a mission, and thus is an important realistic aspect of C^2 systems use.

A certain amount of time-dependent user response can be attained, for example, by including dynamic changes in testbed configurations, such as changing the status of friendly forces due to equipment malfunctions, and enemy forces due to the arrival of new intelligence indicating their current deployment. For example, a flashing icon might alert the user to a newly discovered scud missile site. These kinds of changes will be issued asynchronously by the testbed, so that they appear "real-time" to the user.

This approach allows us to stress the interactions that allow a user to access a situation and allocate appropriate resources. With this approach, we can defer the complicated job of constructing a computer simulation of dynamic, interacting weapons systems composed of sensors, launch platforms, and missiles, at least in the near future. Design of early versions of the testbed will take into account the need for this sort of expansion, to full simulation capabilities, in future versions of the testbed.

2.2.6. Design Issues

The first version of the testbed, in accordance with our "keep it simple" philosophy, will incorporate a map of some militarily "interesting" geographic area of the world that also has interesting shoreline/ocean features (e.g., Middle East, Libya, Korea). The scene will also contain objects of Naval interest (e.g., ships, air bases, subs, sonar, radar) for which information can be accessed by the user.

We have decided at this point to have pre-screen projection be the first new interaction technique in the testbed. We want to be sure that we can get interesting results from our earliest experiments using the testbed, and eye gaze technology may not be mature enough to do this.

Important generic tasks in C^2 systems that will be used as the basis for early testbeds include:

- Acquiring information
- Using the acquired information
- Controlling the display (especially the kind and amount of information presented)

There are several fundamental questions that seem to be most important to explore first for performing these tasks with pre-screen projection:

- What amount of perspective scaling control of the image is appropriate for each of the x, y, and z axes along which changes occur as a user's head moves. This needs to be user-controllable, depending upon what task a user is performing. It also depends upon depth versus height versus width (x, y, z) of the world view available to a user at any point in time. For example, as a user wants "deeper" (more detailed) information, say, about an object on the screen, they may need to work within a small range of the z axis, and within this small range may need greater magnification than when working through the entire z axis range. We will have user-controlled warping in the first testbed.

- How to freeze an image on the screen (it seems unlikely that a user will accept a constantly changing image in response to continual head movements of a user). Initially we will probably use a mouse click to do this; later, gestures or postures from a data glove or words from a voice recognizer, for example, will be included.

• How to unfreeze an image on the screen. This also includes what happens to the screen image when a user issues an unfreeze command. In particular, the view of the screen immediately after unfreezing it can be either absolute or relative to its initial position. We will have both relative and absolute unfreezing in the first testbed.

• How to show increasingly more details of data about a single object. As the user moves closer to the screen, more data about an object are displayed, and vice versa. Hill and Hollan (1992) demonstrated the effectiveness of displaying successively more detailed levels of information as a user zooms in on a scene, using hand controls. We will incorporate this facility into our C^2 version of pre-screen projection. We will explore how best to display simple linear data, as well as hierarchical data (where the user can progressively request more detailed information). We will have ways of displaying linear data in the first testbed, and possibly hierarchical data as well.

• How to show information about multiple objects on the screen at once. In almost any complex situation, such as C^2 systems afford, a user will want to access information about more than one object at a time. For example, in planning a targeting strategy, a user may need to see which weapons are on each of several ships in order to develop an effective targeting plan. We are not sure yet if this will be in the first testbed.

• What sort of context (if any) a user needs when using pre-screen projection. Because a user can always easily back out to the "big picture", the need for a context may be reduced with this interaction technique. We are not sure if this will be in the first testbed.

All these questions involve designing scenarios for appropriate tasks; this is one of our next endeavors. The issues that we will directly address in the first version of the testbed will be (more or less in the order listed, because this seems like the order in which we logically need to know about each of these):

• Perspective scaling control by the user

• Disclosure of simple information about a single object

• Freeze/unfreeze (absolute and relative)

To follow shortly in later versions of the testbed will be:

• Disclosure of hierarchical information about a single object

• Disclosure of information about multiple objects

• Context information

We will develop specific scenarios and tasks that allow us to informally, formatively evaluate these issues. We will then have some (at least loosely acquired) empirical information about design with pre-screen projection. We already have many ideas for designing in order to address

these issues. For example, we may explore the possibility of attaching a trackball to the arm of the user's chair, as a convenient placement for this device.

We may also do some simple comparative studies of pre-screen projection versus traditional techniques (probably mouse and/or trackball) using the early scenarios. As formative evaluation progresses, we will be using our results to design a more interesting, more realistic multi-modal testbed, with which more useful comparative empirical studies can be performed. Important data to collect for pre-screen projection include user performance (e.g., time, errors), user fatigue, and user subjective satisfaction. Use of the technique while standing versus sitting may also be explored.

There are some commercially available tools, such as HyperCard and other similar packages, that can be used for prototyping user interfaces. Unfortunately, there are not commercially available tools that will support implementation of the kinds of novel interaction techniques we are developing. There is simply no way of implementing pre-screen projection or eye-gaze selection using these tools, so we are building our own support tools.

2.3. Extensible User Interfaces

A second important reason for placing the new interaction techniques within a quasi-realistic application is that it allows us to examine the relationship between the interaction software and the rest of the application software. There needs to be a way of adding and refining interaction techniques to an application without having to constantly rewrite major sections of the code each time. In many existing systems, code used to carry out interaction techniques is threaded throughout the application, locking in the way things are handled. It is very difficult, for example, to substitute one means of selecting an object for another. This is not a problem if two input devices, say a mouse and a trackball, are used in virtually exactly the same way; then one device easily substitutes for the other. But it is a problem when different devices operate in different ways to carry out the same task. Our eye-tracking system allows the user to select an object on the screen, but it is performed in quite a different way than it would be using a mouse.

As the variety and combinations of input devices and techniques grows, it becomes more and more important to manage the complexity of programming their use. Ideally, one should be able to plug in a new device, select the tasks it will allow users to perform, and specify how it will be used (the interaction technique), and the application should extend to support it. A carefully designed software architecture is required to approach this level of extensibility.

2.3.1. The User Interface Management System

Interactive computer graphics systems can become large, complex, and difficult to program. In the process of developing the new techniques and testbed, we are also extending the software architecture used to implement the system. We are working in the software constructional domain, specifying what the computer does, but at a higher level of abstraction than writing code. The software to support interactive graphics is constructed within a general purpose framework called a user interface management system, or UIMS.

A UIMS is a separate software component that conducts all interactions with the user; it is separate from the application program that performs the underlying task. It is analogous to a database management system in that it separates a function used by many applications and moves it to a shared subsystem. It removes the problem of programming the user interface from each individual application and permits some of the effort of implementing human-computer interaction to be amortized over many applications and shared by them. It also encourages consistent "look and feel" in user interfaces to different systems, since they share the user interface component. Conversely, it permits human-computer dialogue independence (Hix & Hartson 1993), where changes can be made to an interface design without affecting application code. A UIMS that includes a method for precisely specifying user interfaces allows an interface designer to describe and study a variety of possible user interfaces, such as performing predictive analytic evaluation of an interface before building it.

The UIMS being extended and applied to implement the C^2 testbed was originally developed at the HCI lab (Jacob 1985, 1986). It partitions each interaction task into three levels: lexical, syntactic, and semantic. The *lexical* level covers all physical actions performed by the user and all the low-level processing used to convert that action into internal data that a computer can operate on. It also includes output actions performed by the computer, like drawing a rectangle on the screen. Discrete actions performed either by the user or the computer are typically referred to as events, and a record describing an event is called a token. All input tokens are entered into a common event list. At the *syntactic* level this information fits into the ongoing interaction between user and computer. It determines what user inputs are accepted under the current state of the system, and how the computer reacts to a user's input. It specifies the relationships among events. This level often includes direct, low-level feedback, like highlighting a selected object on the screen. Syntactic level behavior is performed by executing lexical actions. Finally, there is the *semantic* level of interaction – what does a successfully completed sequence of user input mean? This usually translates into the effect user input has on an application. That input might request information, manipulate objects in the application, or enter new data. This flexible three-tiered decomposition of interaction was originally described by Foley and Wallace (1974) as a general way of characterizing a user interface. We have found that it can be applied to a wide variety of

interaction techniques in a straightforward manner. This logical decomposition describes the software architecture we use to implement interaction techniques and how we link them into an application. It is shown in Figure 2.

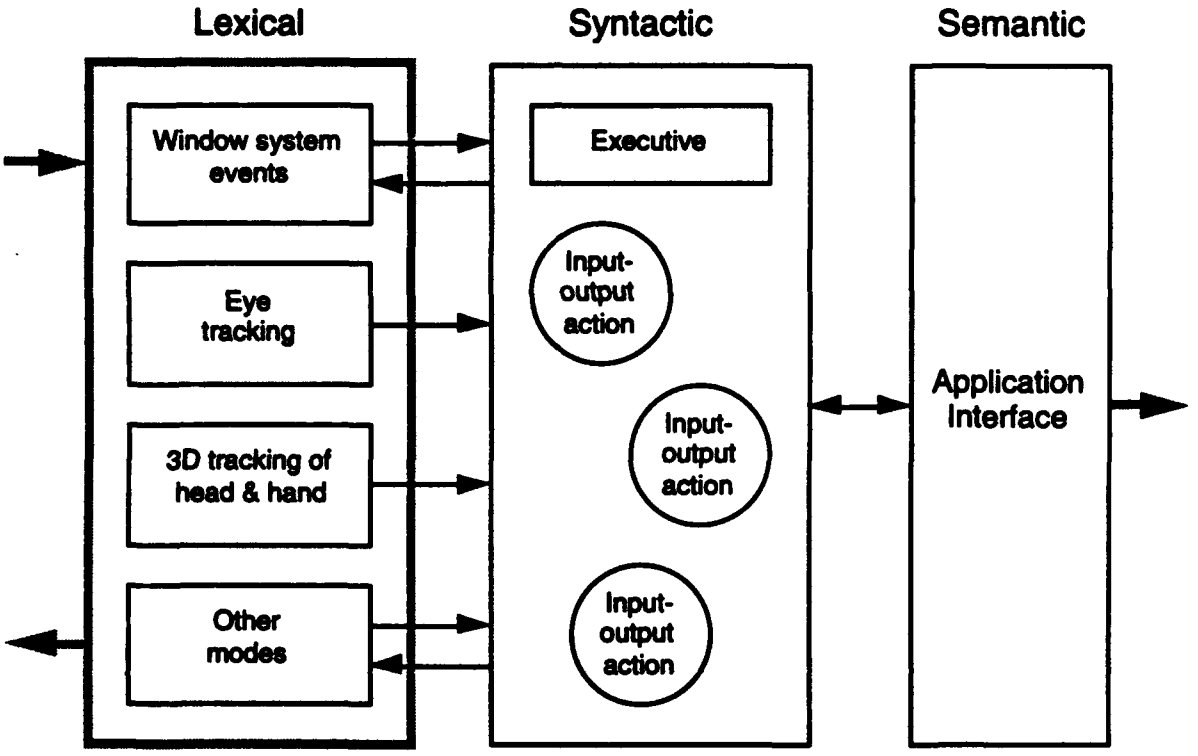

FIGURE 2: DIVIDING THE USER INTERFACE INTO THREE LEVELS: THE LEXICAL, SYNTACTIC, AND SEMANTIC LEVELS.
Details concerned with the software for reading and controlling a variety of hardware devices are handled in the *lexical* portion of the UIMS. All user input is mapped onto the same stream of tokens. These translated tokens, indicating discrete events, are passed to the *syntactic* section which uses state transition graphs to sequence interactions. Each input-output action is handled by a separate co-routine. The executive within the syntactic portion takes care of suspending and resuming these co-routines. Processed input is passed to the application, and graphical output commands are received from it. The application is the *semantic* part of the interactive system.

2.3.2. A Means of Specifying the Syntax of Interaction

Previous design efforts at the HCI lab have led to development of a state transition diagram based approach for specifying the syntactic component of interaction techniques (Jacob 1985, 1986). State transition diagrams show how the user's actions affect system state, depicting possible sequences of operations a user can perform to carry out a given task. Their graphic form makes them easy to work with and explain to other members of a design team.

This approach has been extended to work with direct manipulation user interfaces. Specific manipulations can be described using self-contained sub-diagrams that interact with each other in limited ways. This is an advantage because it is clearer to separate the diagrams used to specify distinct interactions. In that case, a state transition diagram breaks down neatly into a set of concurrently active sub-diagrams, implemented as a collection of co-routines. This approach structures the design specification in a highly modular way, clearly delimiting the interaction between different sub-components. The novel interaction techniques we have developed fit nicely within this framework.

2.3.3. Implementation

We will give an example to illustrate how an interaction technique is specified and processed in terms of state transition diagrams. As shown in Figure 2, tokens obtained as a result of lexical processing feed into state diagrams that define the syntax of the interaction. At the lexical level, event tokens are time-stamped, and multiplexed into a common stream before being passed on to the syntactic portion of the UIMS (Jacob 1985, 1986). For example, eye tokens are translated into the same stream with those generated by the mouse and keyboard.

The desired user interface is specified to the UIMS as a collection of relatively simple individual interactions, represented by separate *interaction objects,* which comprise the user interface description language. A typical object might be a screen button, scroll bar, text field, or eye-selectable graphic icon. A ship icon that can be selected by looking at it will be used as an example below. At the level of individual objects, each such object conducts only a single-thread interaction, with all inputs serialized and with a remembered state whenever the individual dialogue is interrupted by that of another interaction object. Thus, operation of each interaction object is conveniently specified as a simple single-thread state transition diagram that accepts tokens as input. Each object can accept any combination of eye, mouse, and keyboard tokens, as specified in its own syntax diagram, and provides a standard method that the executive can call to offer it an input token and traverse its diagram. Each interaction object is also capable of redrawing itself; it either contains or has access to the necessary state information. An interaction object can have different screen extents (bounding regions) for redrawing, accepting mouse tokens, and accepting eye tokens.

The executive within the syntactic portion of the UIMS controls the flow of control between high-level state transition diagrams. It operates by collecting all state diagrams of the interaction objects and executing them as a collection of co-routines, assigning input tokens to them and arbitrating among them (e.g., activating and suspending with retained state), as they proceed. Whenever the currently-active state transition diagram receives a token it cannot accept, the

executive causes it to relinquish control by co-routine call to whatever diagram can, given its current state, accept it. If none can, the executive discards the token and proceeds.

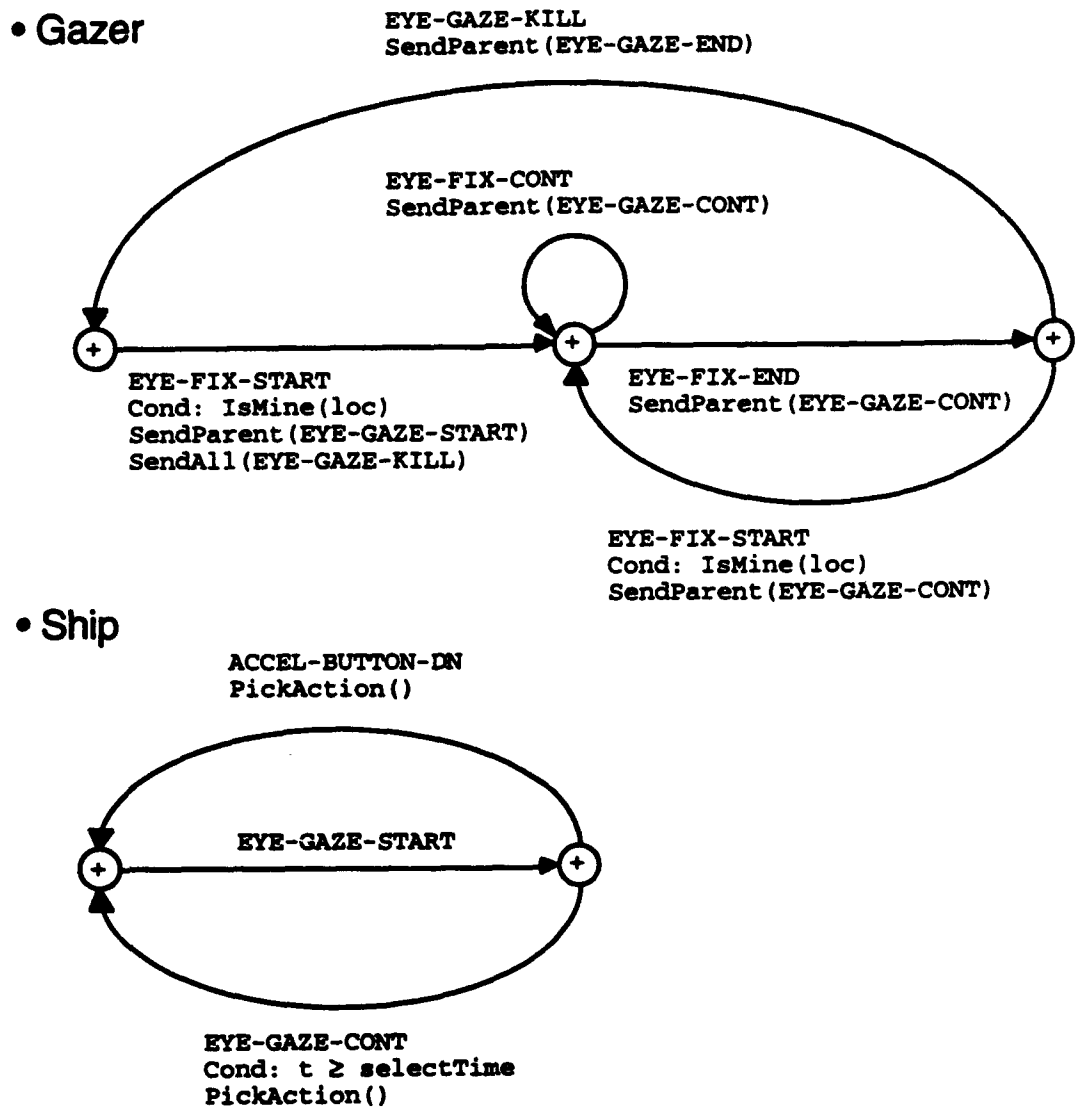

FIGURE 3: STATE TRANSITION DIAGRAMS FOR AN EYE-SELECTABLE OBJECT. Nodes in diagrams represent states in human-computer interaction, waiting for events. Directed arcs represent events triggered by user actions. Upper-case words label events. Expressions which include parentheses indicate function calls. Cond: specifies a condition that must hold before its associated event triggers a state change.

For example in Figure 3, each *Ship* is a separate interaction object (but all are of the same class, Ship). An additional lower-level interaction object, *Gazer*, is provided to translate fixations into

gazes. That is, every interaction object such as Ship also has a Gazer interaction object associated with it. The Gazer accepts fixations on its parent object and then combines such consecutive fixations into a single gaze token, which it sends to its parent object (in this case, the Ship). Figure 3 shows the state transition diagram for Gazer; it accepts tokens generated by the fixation recognition algorithm (EYE-FIX-START, EYE-FIX-CONT, and EYE-FIX-END), tests whether they lie within its extent or else meet the criteria for off-target fixations described above (implemented in the call to IsMine), accumulates them into gazes, and sends gaze tokens (EYE-GAZE-START, EYE-GAZE-CONT, and EYE-GAZE-END) directly to the parent object. The Ship interaction object then need only accept and respond to gaze tokens sent by its associated Gazer object. Figure 3 also shows the portion of the Ship interaction object state transition diagram concerned with selecting a ship by looking at it for a given dwell time (for clarity the syntax for dragging and other operations is not shown in the figure; also not shown are tokens the selected ship sends to other ships to deselect any previously-selected ship). When a user operation on a ship causes a semantic level consequence (e.g., moving a ship changes its track data), the Ship interaction object calls its parent, an application domain object, to perform the work. This syntax is well matched to natural saccades and fixations of the eye. Notice that although the Gazer interaction object describes how the system registers one form of user *input*, and the Ship interaction object describes the *output* response of a displayed item, they can both be clearly represented using the same notation.

3. THE SOFTWARE DEVELOPMENT ENVIRONMENT

New interactive devices and techniques are being invented and commercialized as advanced technology is applied to open up communications channels between humans and computers. We do not want to expend a great deal of effort to develop a software implementation that can only be used with one or two specialized devices. Thus, we are developing a general purpose software architecture that extends easily to support additional new devices and techniques.

Tools and approaches we are using to implement and build our C^2 testbed and underlying software architecture are an important part of our work. Substantial effort must go into an interactive system's design to structure its software into modular functional components. The functionality described here shows how information is handled, elevating user actions from raw input signals to application-significant operations, while providing feedback at one or more levels. Each new technique is decomposed in such a way that code modules required to implement it plug smoothly into the existing testbed system. The portion of the system designed to handle this is the UIMS, described previously in Section 2.3.

3.1. Object-Oriented Design

The C^2 testbed, details of the interaction techniques, and the UIMS are designed using an object-oriented approach. An object-oriented language is well suited for coding interactive graphics systems which can be very difficult to program if their complexity is not managed. (The class hierarchy for the UIMS-based software architecture is given in the Appendix.) Interaction objects described in the preceding section are abstract entities, and should not be confused with objects of the programming language used to implement them.

Interactive computer graphics is a time critical application area. A number of advanced, machine-portable object-oriented programming languages are available, but most of them are not designed to be extremely run-time efficient. Execution speed has often been traded to achieve a pure and flexible object-oriented language like Smalltalk, Eiffel, or CLOS (Common Lisp Object System). C++ was designed from the start to retain as much of the speed advantages of C as possible in an object-oriented system.

Thus we are using the C++ programming language to implement the interaction techniques, the testbed, and the UIMS. Standard libraries that support graphical user interaction were built to be accessed from C, and are therefore available through C++. Excellent code management tools and debugging tools are available on both hardware platforms (Sun and Silicon Graphics, discussed in later sections) we are using.

3.2. Windowing Systems and Graphics Tools

Strengths and weaknesses of existing software tools and development techniques, including commercially available ones such as the X Windows system and Motif, discussed below, become evident only when they are applied to the wide variety of situations encountered as we explore new ways of performing graphical interaction. Our group attempts to leverage existing tools to implement new interaction techniques, but sometimes existing systems are too limited. We have found that much of the functionality provided by the X Window system and nearly all of what Motif provides is not useful for building new interaction techniques that use input devices other than a mouse or keyboard, because these systems were not designed to handle different devices. Furthermore, they are not modular or extensible enough for our purposes. The problems we uncover with these systems are useful discoveries in and of themselves, especially when we can produce better methods.

3.2.1. Strengths and Limitations of the X Window System

The strength of X Windows is that it places an efficient low-level windowing support system within the context of a networked environment. It provides necessary tools for developing

interactive graphics based on mouse and keyboard input devices, and 2D graphical output. Most importantly of all, the X Window system has become a standard means for Unix-based workstations to provide support for interactive graphics.

Windowing systems that rely on mouse input have evolved to the point where a general support package can be provided. They work well with devices other than a mouse, say a trackball or tablet that can perform exactly the same function as a mouse; i.e., they can be used to position the cursor precisely within a 2D plane.

Unfortunately, X is a rather awkward tool for working with novel input devices that do something other than emulate a mouse or keyboard in terms of X level protocols. X only supports these two classes of devices, and only allows them to be used in specific ways. However, with the X Window system we find that a simple subset is useful to us, but that as the elements within a system start to interact, we must take over and manage these facilities ourselves, building a more flexible system as we go. We use basic features of X, but shift the more complex aspects of dynamic screen control to our own software support system. We use Xlib (X Window's low-level function library) for drawing graphics, handling server events, and dealing with mouse input events. We have built our own system on top of Xlib for dealing with novel devices, event queues, screen management, and graphical objects.

Novel devices like the eye tracker and the 3D head tracker interface to graphics workstations over an RS-232 serial interface. This provides direct access to the input device, without having to deal with X protocols which are not set up to handle such devices. The low-level (lexical) software that reads and controls these devices is part of our UIMS (see Section 2.3.1). It would be possible but considerably more difficult to incorporate them inside X itself, because X has a layer structure tailored to work with a limited set of interactive devices. X has its own restrictive way of doing things that extends beyond the way it handles lexical input. Its screen management and tokenized event list is not always suited for our new techniques, so in both cases we only use the most elementary parts and use our own system to manage higher-level relationships.

3.2.2. Why Motif Is Not Used to Construct the Interactive Testbed

The over-specialization of current windowing packages built to facilitate mouse-based interactions extends to higher-level packages that give an implementer a set of widgets and a collection of design rules with which to implement interfaces with a consistent "look and feel". (A widget is a graphical device presented on the screen that a user can operate, such as a menu or scroll-bar. It is a highly stylized interaction technique.) Guidelines for designing WIMP (windows, icons, menus, pointers) systems ignore the possibility of using input devices other than a mouse look-alike and a keyboard.

Each device has its own capabilities and limitations. The visual appearance and action of a workable object on the screen is an important part of an interaction technique. It provides the visual feedback necessary for a user to control a device, and to confirm that the desired operation was successfully completed. Just because an eye tracker and a mouse can both be used to select an object on the screen does not mean that an eye tracker can be substituted for a mouse in an existing application. For example, eye-based selection is faster than using a mouse, but it can only handle targets larger than one degree of visual arc and can be used to select items that are more than two degrees of visual arc apart (Jacob 1991). Thus, many of interaction techniques provided by Motif (via Motif widgets) simply will not work for eye-gaze-based interaction.

3.2.3. GL & X Windows

We use a Silicon Graphics, high-performance workstation to perform the continuous viewing transformations needed by our graphics-intensive interaction techniques, such as the pre-screen projection technique (Section 2.1.1). Silicon Graphics provides a very well designed, efficient, and easy to use graphics library for drawing two and three dimensional pictures on interactive color workstations. In its current form, SGL (Silicon Graphics Library) includes basic window and input management functions that cover much of the same ground as Xlib. Until now, SGL has been a proprietary development tool, only available on Silicon Graphics hardware.

The X Window system, on the other hand, has become a defacto standard for portable window-based software development, and is available on a wide variety of computer systems, workstations, and terminals. It has limitations in that it supports only 2D graphics primitives, and its range of input devices is limited, as explained in Section 3.2.1. The basic X operations, like specifying, opening, and redrawing a window, are generally useful commands. The fact that they can work over a distributed network contributes to their market penetration.

There is a current movement in the computer graphics industry to provide a portable, SGL-like graphics support library that operates within the framework of X. Within their own product line, Silicon Graphics is moving SGL away from a full feature interactive graphics support system and repackaging it as a set of output-only rendering routines. Windowing and input management routines are being stripped away from SGL, leaving only rendering components that draw images within windows defined by X. At the same time, a new graphics library called Open-GL is being developed as an industry standard software package. Open-GL is fashioned after SGL, and like the forthcoming version of SGL, it is designed to work within an X Window system "wrapper".

This fits nicely with the portable software architecture we are developing. SGL will provide 3D graphics rendering. In the future it will be displayed within an X window and SGL will be supplanted by Open-GL. Only a minimal subset of X will be used. The majority of input device

control, event queue management, overall screen layout, and interaction techniques will be handled by our own UIMS system.

3.3. Multi-Platform Development

The UIMS and application testbed are both being developed to run across several platforms. Although this increases the effort involved in initial implementation, it ensures that the resulting software is not tied to a single computer hardware platform.

3.3.1. Sun for Eye Tracking

Our eye tracking system consists of a corneal reflection eye tracking unit connected to a Sun computer workstation. The eye tracking hardware is an Applied Science Laboratories Model 4250R system, consisting of an optical system that illuminates the eye with infrared light and senses the eye with a video camera. The eye tracker is controlled by an Intel 486-based computer which performs feature recognition on the video image to determine where the eye is looking. This subsystem transmits the x and y coordinates designating the line of gaze over a serial port once every sixtieth of a second to the Sun computer.

The Sun performs all further processing, filtering, fixation recognition, and some additional calibration. Software on the Sun written in C++ parses the raw eye tracker data streams into tokens that represent meaningful input events. The UIMS deals with queuing these events, feeding them to the state-transition-based parser to implement the pre-defined syntax of eye-driven interaction techniques.

Currently all graphics drawing on the Sun is performed using X. (Open-GL is not yet available for the Sun.) Text and 2D graphics are used to present a static display of ships. When the user looks at a particular ship, its image highlights, and text appears to the side of the screen, presenting additional information about the ship. Both pop-up and constantly displayed menus are implemented to provide access to additional information about each ship. The UIMS couples the user's actions to the display and allows direct manipulation of application-specific information.

3.3.2. Silicon Graphics for Pre-Screen Projection

The high-performance 3D graphics pipeline on the Silicon Graphics workstation is required to support the continually changing view presented with pre-screen projection. The initial version of this technique was quickly coded using SGL to test out the basic premise that it is natural to control imagery that moves in response to a user's head motions as if it were located in front of the screen (see Section 2.1.1).

We have connected two different 3D tracking devices to our Silicon Graphics workstation: an Ascension Bird™, and a Polhemus 3-Space tracking device. Both track and report the location (x, y, z) and orientation (yaw, pitch, roll) of their sensor. The pre-screen projection technique only makes use of position information and ignores the orientation of a user's head. A continuous stream of coordinates is reported to the host computer. The Silicon Graphics constantly redraws the view of the map on the screen to keep it aligned with the user's head position.

3.3.2.1. Dynamic 3D Rendering in Pre-Screen Projection

Our C^2-like testbed consists of a variety of ships and planes displayed on a geographical map. The way a map is processed for display illustrates the challenges associated with using new interaction techniques within the context of a complex application. The changing views presented in response to the user's head motion must be presented in quick succession to produce a smooth visual transformation. The more frames per second are drawn, the smoother the animation becomes. Thus the map should be redrawn as quickly as possible. Commercially available maps portray geographic and political boundaries with a high degree of accuracy. A large number of edge coordinates must be processed by the graphics hardware to draw such a map on the screen. Of course, it takes longer to draw shapes with a large number of edges than those with a few. The longer it takes to draw an individual picture frame, the fewer frames are drawn per second, and the more jerky the continuously changing view becomes.

We want to evaluate pre-screen projection under the best available conditions. By manually digitizing a map, we can keep the number of edge coordinates to a minimum and concentrate on details of the map in areas of interest for the C^2 scenarios. A map of North Africa and the Mediterranean has been digitized manually to allow pre-screen projection to perform as smoothly as possible. It is essential to optimize performance of a new interaction technique so that irrelevant hardware and software limitations can be factored out of evaluation of the technique within the context of the testbed (i.e., frame generation rate should not be a factor in evaluation).

This illustrates the point that whenever a new technique is developed, a new set of issues arise that effect the way other parts of the testbed operate. This leads to a trade-off between maintaining control over details of how an interaction technique works, and making do with more readily available methods for implementing the technique. In the case of the map, a high precision geographic contour map could be used, as is typically used when a map is zoomed in and out by large steps. But redrawing so much detail would slow the frame rate at which pre-screen projection imagery is presented. The alternative is to change the overall level of detail in the map to make pre-screen projection work as smoothly as possible on our hardware. These sorts of issues occur repeatedly as we fill in the practical details of each new technique. Extra work is often

required to tailor the way other parts of a user interface work with new techniques. Just as, historically, a great deal of effort has already gone into creating and tuning mouse-based windowing systems, each new technique must be carefully tuned to fit its appropriate context. Only then can it be fairly evaluated and compared against existing techniques.

A goal of our current effort is to display a sufficiently interesting scene of a Naval situation to allow us refine details of pre-screen projection and evaluate it based on actual use. The display will include ships, planes, airfields, and military targets displayed on a map. Each object will be labeled, and techniques will be provided, for reference, to present status information about each object or group of objects. This kind of information is essential for performing mission planning tasks in the C^2 testbed.

3.3.3. Compatibility Issues

The current version of the UIMS runs on a Sun and uses the X Window system for its low-level graphical input and output. Before the code for pre-screen projection can be integrated into the UIMS, it must be compatible with X. Therefore, once a basic version of the testbed is working, the next step will be to convert the software running on the Silicon Graphics to make it compatible with X. That will require converting the windowing and input event management from SGL to X Windows. This coding task must be done all at once, since there is no way to mix events reported by SGL with windows managed by X. How the two systems initialize the screen and handle color are also incompatible.

This step paves the way for transporting the UIMS, which currently only runs on the Sun, to the Silicon Graphics. The code should easily port to the Silicon Graphics machine, since it was developed under Unix and written primarily in C++. The only software that is specific to the Sun is the code used to drive the eye tracker.

There are differences between the versions of Unix running on the two computers, in particular the "Make" code management facility on the Silicon Graphics is not as sophisticated as that of the Sun's. These differences should be easily surmounted as the UIMS is transported to the Silicon Graphics. It is interesting to note that one of the most difficult aspects of developing software on two different machines is that each machine provides its own set of development tools (e.g., editors, debuggers, code version management, etc.). Thus a developer skilled in using one development environment may not be as proficient when using the other.

4. SUMMARY

At the Naval Research Laboratory's Human-Computer Interaction Laboratory, we are developing novel interaction techniques to facilitate graphical interaction with command and control, and other interactive, systems. In one technique, eye gazes are tracked and used to select items to present more information about them. The other technique, pre-screen projection, a newly invented technique for panning and zooming over a scene, appears well suited for facilitating the kinds of spatial resource management carried out in command and control applications. Once fully developed, this should facilitate analysts' use of situational displays presented on desktop or large screen displays.

We have begun developing a command and control application testbed which will provide a task-oriented context in which to empirically evaluate the new interaction techniques. The testbed will be built to include the interaction techniques described in this report, alone and in combination with other techniques. Requirements for the application testbed have been carefully studied, to limit it to a manageable development task, and implementation of the testbed is in the early stages.

As discussed, our UIMS is capable of easily dealing with new devices and interaction techniques as we add them to the testbed. Further efforts will explore how well the UIMS helps us hook a variety of interaction techniques into the application testbed. If this software architecture continues to prove as useful and flexible as it has been until now, it will serve as a model for constructing future applications and allow them to use a wider range of input devices. This approach should apply not just to Naval interactive systems, but to scientific visualization and virtual reality based systems as well.

ACKNOWLEDGMENTS

The authors wish to acknowledge Ms. Linda Sibert and Dr. Helen Gigley, both of the NRL HCI Laboratory, for their editorial reviews. This work was sponsored by Decision Support Technology block (RL2C) within the ONR Exploratory Development Program, managed by Elizabeth Wald.

REFERENCES

Deering, M. (1992) High resolution virtual reality. *Computer Graphics* (ACM) v 26, n 2, pp. 195-202.

Foley, J.D. and V.L. Wallace. (1974) The Art of Natural Graphic Man-Machine Conversation. *Proc. IEEE* v 64, n 4, pp. 462-470.

Hill, W. C., & Hollan, J. D. (1992). Pointing and visualization. *Proc. ACM CHI'92 Human Factors in Computing Systems*, pp. 665-666.

Hix, D. (1993) "A User Task Analysis for Command and Control Systems and Its Use in Human-Computer Interaction Research," Naval Research Laboratory Report: NRL/MR/5530-93-7397.

Hix, D., & Hartson, H. R. (1993). *Developing user interfaces: Ensuring usability through product & process.* John Wiley & Sons, Inc., New York, NY.

Jacob, R.J.K. (1985) "A State Transition Diagram Language for Visual Programming," *IEEE Computer*, n 18, n 8, pp. 51-59.

Jacob, R.J.K. (1986) "A Specification Language for Direct Manipulation User Interfaces," Special Issue on User Interface Software of *ACM Transactions on Graphics*, v 5, n 4, pp. 283-317.

Jacob, R.J.K. (1990) "What You Look At is What You Get: Eye Movement-Based Interaction Techniques," *Proc. ACM CHI'90 Human Factors in Computing Systems Conference*, pp. 11-18, Addison-Wesley/ACM Press.

Jacob, R.J.K. (1991) "The Use of Eye Movements in Human- Computer Interaction Techniques: What You Look At is What You Get," *ACM Transactions on Information Systems*, v 9, n3, pp. 152-169.

Templeman, J. (1993a) Extending Virtual Reality Out In Front of the Screen. *Virtual Reality Systems*, v 2, n 1 - in Press.

Templeman, J. (1993b) Pre-screen Projection of Virtual Scenes. *Virtual Reality Conference Proceedings.* Fall'93 - in Press.